```cpp
#include <Windows.h>
#include "Level.h"
#include <iostream>
#include <fstream>
#include "Player.h"
#include "Enemy.h" // derived Placeable Actor Classes
#include "Key.h"
#include "Door.h"
#include "Goal.h"
#include "Money.h"
#include <assert.h>

using namespace std;

constexpr char WAL = (char)219;

Level::Level()
    : plevel(nullptr)
    , height(0)
    , width(0)
{

};

Level::~Level()
{
    if (plevel != nullptr)
    {
        delete[] plevel;
        plevel = nullptr;
    }

    while (!m_pActors.empty())
    {
        delete m_pActors.back(); // return us the elements at end, then delete
        m_pActors.pop_back(); // continue to delete the remaining vector elements.
    }
};

bool Level::LoadLevel(string levelName, int* playerX, int* playerY)
{
    levelName.insert(0, "../");
    ifstream levelFile;
    levelFile.open(levelName);
    if (!levelFile)
    {
        cout << "An error has occured." << endl;
        return false;
    }
    else
    {
```

```
52          constexpr int tempSize = 25;
53          char temp[tempSize];
54
55          levelFile.getline(temp, tempSize, '\n');
56          width = atoi(temp); // converts integer into width.
57
58          levelFile.getline(temp, tempSize, '\n');
59          height = atoi(temp);
60
61          plevel = new char[width * height]; // array that we need to
              deallocate.
62          levelFile.read(plevel, width * height);
63
64          if (playerX != nullptr && playerY != nullptr)
65          {
66              bool anyWarnings = Convert(playerX, playerY);
67              if (anyWarnings)
68              {
69                  cout << "There are some warnings in the level data. see
                      above." << endl;
70                  system("pause");
71              }
72          }
73          return true;
74      }
75  }
76
77  void Level::Draw()
78  {
79      HANDLE console = GetStdHandle(STD_OUTPUT_HANDLE);
80      SetConsoleTextAttribute(console, kRegularColour);
81
82      //Draw Level
83      for (int y = 0; y < GetHeight(); ++y)
84      {
85          for (int x = 0; x < GetWidth(); ++x)
86          {
87              int indexToPoint = GetIndex(x, y);
88              cout << plevel[indexToPoint];
89          }
90          cout << endl;
91      }
92
93      COORD actorCursorPosition; // position the cursor at correct location, x
           and y variables
94
95      // Draw actors
96
97      for (auto actor = m_pActors.begin(); actor != m_pActors.end(); +
          +actor) // going to the beginning and through the end.
98      {
99          if ((*actor)->IsActive()) // if active we want to draw.
100         {
```

```
101             actorCursorPosition.X = (*actor)->GetXPosition();
102             actorCursorPosition.Y = (*actor)->GetYPosition();
103             SetConsoleCursorPosition(console, actorCursorPosition); // set
                  position
104             (*actor)->Draw(); // draw the actors
105         }
106     }
107 }
108
109 bool Level::IsSpace(int x, int y)
110 {
111     return plevel[GetIndex(x, y)] == ' ';
112 }
113 bool Level::IsWall(int x, int y)
114 {
115     return plevel[GetIndex(x, y)] == WAL;
116 }
117
118 bool Level::Convert(int* playerX, int* playerY)
119 {
120     bool anyWarnings = false;
121
122     for (int y = 0; y < height; ++y)
123     {
124         for (int x = 0; x < width; ++x)
125         {
126             int intIndex = GetIndex(x, y);
127
128             switch (plevel[intIndex])
129             {
130             case '+':
131             case '-':
132             case '|':
133             {
134                 plevel[intIndex] = WAL;
135                 break;
136             }
137             case ' ':
138             {
139                 break;
140             };
141             case 'r':
142                 plevel[intIndex] = ' ';
143                 m_pActors.push_back(new Key(x, y, kRedColour));
144                 break;
145             case 'g':
146                 plevel[intIndex] = ' ';
147                 m_pActors.push_back(new Key(x, y, kGreenColour));
148                 break;
149             case 'b':
150                 plevel[intIndex] = ' ';
151                 m_pActors.push_back(new Key(x, y, kBlueColour));
152                 break;
```

```
153             case 'R':
154                 plevel[intIndex] = ' ';
155                 m_pActors.push_back(new Door(x, y, kRedColour,
                      kRedColourSolid));
156                 break;
157             case 'G':
158                 plevel[intIndex] = ' ';
159                 m_pActors.push_back(new Door(x, y, kGreenColour,
                      kGreenColourSolid));
160                 break;
161             case 'B':
162                 plevel[intIndex] = ' ';
163                 m_pActors.push_back(new Door(x, y, kBlueColour,
                      kBlueColourSolid));
164                 break;
165             case 'X':
166                 plevel[intIndex] = ' ';
167                 m_pActors.push_back(new Goal(x, y));
168                 break;
169             case '$':
170                 plevel[intIndex] = ' ';
171                 m_pActors.push_back(new Money(x, y, 1 + rand() % 5));
172                 break;
173             case '@':
174             {
175                 plevel[intIndex] = ' ';
176                 if (playerX != nullptr && playerY != nullptr)
177                 {
178                     *playerX = x;
179                     *playerY = y;
180                 }
181                 break;
182             }
183             case 'e':
184                 m_pActors.push_back(new Enemy(x, y));
185                 plevel[intIndex] = ' '; // clear level
186                 break;
187             case 'h': // horiztonal enemy
188                 m_pActors.push_back(new Enemy(x, y, 3, 0));
189                 plevel[intIndex] = ' ';
190                 break;
191             case 'v': // vertical enemy
192                 plevel[intIndex] = ' ';
193                 m_pActors.push_back(new Enemy(x, y, 0, 2));
194                 plevel[intIndex] = ' ';
195                 break;
196             default:
197             {
198                 cout << "Invalid character in file " << plevel[intIndex] <<
                      endl;
199                 anyWarnings = true;
200                 break;
201             }
```

```
202             }
203         }
204     }
205     return anyWarnings;
206 }
207
208 int Level::GetIndex(int x, int y)
209 {
210     return x + y * width;
211 }
212
213 // Updates all actors and returns a colliding actor is there is one.
214
215 PlaceableActor* Level::UpdateActors(int x, int y) // pass in x and y of
      player.
216 {
217
218     PlaceableActor* collidedActor = nullptr;
219
220     for (auto actor = m_pActors.begin(); actor != m_pActors.end(); ++actor)
221     {
222         (*actor)->Update(); //update all actors
223
224         if (x == (*actor)->GetXPosition() && y == (*actor)->GetYPosition
              ()) // collision occured
225         {
226             assert(collidedActor == nullptr); // cannot collide with
                  multiple actor
227             collidedActor = (*actor); // collided actor to the actors x and
                  y
228         }
229     }
230     return collidedActor;
231 }
```